EECS 3101 M: Design and Analysis of Algorithms

Suprakash Datta
Office: LAS 3043

Course page: `http://www.eecs.yorku.ca/course/3101M`
Also on Moodle

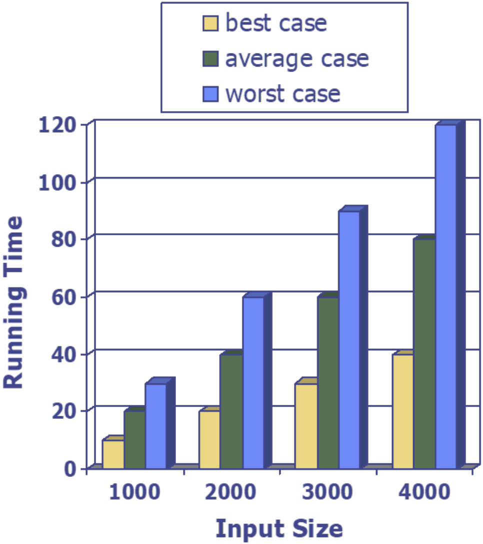
Analysis of FindMax

```
FIND-MAX(A)
1  max = A[1]
2  for j = 2 to A.length
3        if max < A[j]
4        max = A[j]
5  return max
```

line	Cost	Times
1	c_1	1
2	c_2	n
3	c_3	$n-1$
4	c_4	$0 \leq k \leq n-1$
5	c_5	1

Best Case: $k = 0$
Worst Case: $k = n - 1$
Average Case: ?

Best/Worst/Average Case Analysis



Best/Worst/Average Case Analysis - 2

The running time of an algorithm typically grows with the input size.

- Best Case: Not very informative
- Average Case: Often very useful, but hard to determine
- Worst Case: Easier to analyze. Crucial in applications like
  - Games
  - Finance
  - Robotics

Analysis of FindMax - Continued

FIND-MAX(A)

1 $max = A[1]$
2 **for** $j = 2$ **to** $A.length$
3 **if** $max < A[j]$
4   $max = A[j]$
5 **return** max

line	Cost	Times
1	c_1	1
2	c_2	n
3	c_3	$n-1$
4	c_4	$0 \leq k \leq n-1$
5	c_5	1

Running time (worst-case): $c_1 + c_5 - c_3 - c_4 + (c_2 + c_3 + c_4)n$
Running time (best-case): $c_1 + c_5 - c_3 + (c_2 + c_3)n$

Simplifying Running Times

Note that the worst-case time of $c_1 + c_5 - c_3 - c_4 + (c_2 + c_3 + c_4)n$ is

- Complex
- Not useful as the c_i's are machine dependent

A simpler expression: $C + Dn$ [still complex].

Want to say this is Linear, i.e., $\approx n$

Q: How/why can we throw away the coefficient D and the lower order term C?

Simplifying Running Times - Rationale

- Discarding lower order terms: We are interested in large n – cleaner theory, usually realistic.
- Discarding coefficients (multiplicative constants): the coefficients are machine dependent

Caveat: remember these assumptions when interpreting results! We will not get:

- Exact run times
- Comparison for small instances
- Small differences in performance

Analysis of FindMax - Summary

- Last expression: $C + Dn$ written as $\Theta(n)$
- Also called linear time
- Question: Can we do better?

Approach: reason about the **problem** not the algorithm;
show that **any** algorithm for the problem must have worst case running time $\Omega(n)$
i.e.
for any algorithm for this problem, for each $n > 0$, there exists an input that make the algorithm take $\Omega(n)$ time

Lower Bounds

- Consider only comparison-based algorithms; show any such algorithm must use $\Omega(n)$ comparisons in the worst case
- Note that the number of comparisons is a lower bound on the running time of an algorithm
- Warning: we must reason about all algorithms, so we have to be careful not to assume anything about how the algorithm proceeds

Proof of Lower Bound

- Claim: Any comparison-based algorithm for finding the maximum of n *distinct* elements must use at least $n-1$ comparisons.

- Proof:
  If x, y are compared and $x > y$, call x the winner, y the loser.
  Any key that is not the maximum must have lost at least one comparison. $\boxed{\text{WHY?}}$
  Each comparison produces exactly one loser and at most one NEW loser.
  Therefore, at least $n-1$ comparisons have to be made.

Observations

We proved a claim about ANY algorithm that only uses comparisons to find the maximum. Specifically, we made no assumptions about

- Nature of algorithm
- Order or number of comparisons
- Optimality of algorithm
- Whether the algorithm is reasonable – e.g. it could be a very wasteful algorithm, repeating the same comparisons